In [ ]:
```python
import numpy
import numpy as np
import math
import pandas as pd
from scipy.optimize import minimize
import itertools
from functools import reduce
from gt_design import *
```

Define Ranges

Define the range for the stage loading coefficient, flow coefficient and the reaction. Using the obtained values, calculate the other values.

Use the following data:

1. Reaction (Λ) -> 0.35 - 0.65
2. Flow Coefficient (ϕ) -> 0.6 - 1.0
3. Structural Limit (AN^2) -> 10000000 - 22580600 (checking for a wide range)
4. Zweifel for vane -> 0.75 - 0.90
5. Zweifel for rotor -> 0.80 - 0.95

In [ ]:
```python
range_stage_loading = numpy.linspace(2.5, 4.5, 15)
range_alpha_3 = numpy.linspace(10, 25, 10)
range_mach_exit = numpy.linspace(0.3, 0.55, 10)
range_AN_squared = numpy.linspace(15000000, 22580600, 20)
range_zweifel_vane = numpy.linspace(0.75, 0.90, 4)
range_zweifel_rotor = numpy.linspace(0.80, 0.95, 4)
range_incidence = numpy.linspace(2, 4, 6)
```

int main(){}

In [ ]:
```python
def int_main(range_mach_exit, range_stage_loading, range_alpha_3, range_AN_squared, range_incidence, range_zweifel_vane, range_zweifel_roto
    """
    Design Point Angles:
    beta_2m -> rotor inlet metal angle
    beta_3m -> rotor exit metal angle

    Off Design Angles:
    beta_2 -> rotor inlet flow angle
    beta_3 -> rotor exit flow angle

    IMP -> FOR DESIGN POINT -> beta_2m = beta_2, beta_3m -> beta_3

    INCIDENCE -> incidence = beta_2 - beta_2m, DEVIATION (for the project) -> beta_3 = beta_3m
    """
    data_meanline = []
    data_root_hub = []
    data_meanline_losses = []
    data_off_design = []
    data_efficiency = []
    data_scratch = []

    #for i, j, k, l, m, n, o, p in itertools.product(range_mach_exit, range_stage_loading, range_alpha_3, range_degree_reaction, range_AN_sq
    for i, j, k, m, n, o, p in itertools.product(range_mach_exit, range_stage_loading, range_alpha_3, range_AN_squared, range_incidence, ra
        incidence = n

        # MEANLINE ANALYSIS
        T_1, P_1, rho_1, C_1 = aeroturbine.calc_properties(M_1, T_01, P_01)
        C_a_1 = C_1 * np.cos(np.radians(-10))
        T_3, P_3, rho_3, C_3 = aeroturbine.calc_properties(i, T_03_cooled, P_03)
        C_w_1 = numpy.sqrt(C_1**2 - C_a_1**2)

        # Calculate U from the stage loading.
        U = aeroturbine.calc_U(j)
        C_a_3, C_w_3,  V_3, V_w_3, flow_coefficient_3, beta_3m, a_3, M_3_rel, A_3,P_03_rel = aeroturbine.calc_stage_3(U, C_3, T_3, rho_3,P_
        T_02,T_2, P_2, rho_2, A_2, C_a_2, flow_coefficient_2, a_2, V_w_2, beta_2m, V_2, C_w_2, C_2, alpha_2, M_2, M_2_rel,P_02, P_02_rel,l
        if T_02 > 0:

            # STRUCTURAL ANALYSIS
            A_1 = m_dot_1/(rho_1 * C_a_1)
            N, omega, r_hub, r_tip, r_meanline, h = aerostructural.calc_structural(m, A_2, U)

            if V_3 > V_2 and C_2 > C_1 and rho_3 < rho_2 < rho_1 and T_02 < T_01 and T_02 > T_03 and T_1 > T_2 and T_2 > T_3 and P_1 > P_2 

                # BLADE VORTEX ANALYSIS
                alpha_2_hub, alpha_3_hub, beta_2_hub, beta_3_hub, U_hub, V_2_hub, C_2_hub, M_2_rel_hub, M_2_hub, reaction_hub = aeroturbine
                alpha_2_tip, alpha_3_tip, beta_2_tip, beta_3_tip, U_tip, V_2_tip, C_2_tip, M_2_rel_tip, M_2_tip, M_3_rel_tip = aeroturbine.
                #incerased alpha_2 to 73 from 70
                if 40 < alpha_2 < 75  and U_hub < 335.28  and reaction_hub > 0 and M_2_rel_hub > M_2_rel_tip and 3150 < omega < 4500 and P_
                # ANGLES CHECK
                    if 0 < alpha_2_tip < alpha_2 < alpha_2_hub and 0 < beta_2_tip < beta_2m < beta_2_hub and 0 < alpha_3_tip < k < alpha_3_

                        # OFF DESIGN CALCULATIONS
                        T_3_od,rho_3_od, P_3_od, alpha_3_od, beta_2_od, flow_coeff_2_od, incidence_2, v_2_od, C_w_3_od,C_a_3_od,U_mean_od,f
```

```python
                        if flow_coeff_3_od > flow_coeff_2_od and C_a_3_od > 0 and 10 <= alpha_3_od <= 40 and P_02_rel_od > P_03_rel_od:

                            # VANE AREA AND HEIGHT
                            A_vane_mean = (A_1 + A_2)/2
                            h_vane_mean = (A_vane_mean * N/60)/U
                            r_tip_stator = r_meanline + (h_vane_mean/2)
                            r_hub_stator = r_meanline - (h_vane_mean/2)

                            # DESIGN POINT LOSSES
                            # CALCULATE LOSSES - Stator
                            K_p_stator, pitch_chord_ratio_stator, K_accel_stator, stagger_angle_stator, pitch_chord_ratio_stator, pitch_axi
                            K_s_stator = aerodynamic_losses.secondary_losses.calc_K_s(K_accel_stator, AR_vane, -10, alpha_2)
                            K_TET_stator, N_stator, c_true_stator, c_a_stator,throat_opening_stator = aerodynamic_losses.trailing_edge_loss
                            K_stator = K_p_stator + K_s_stator + K_TET_stator
                            pitch_stator = pitch_chord_ratio_stator*c_true_stator

                            # CALCULATE LOSSES - Rotor
                            K_p_rotor, pitch_chord_ratio_rotor, K_accel_rotor, stagger_angle_rotor, pitch_chord_ratio_rotor, pitch_axial_ch
                            K_s_rotor = aerodynamic_losses.secondary_losses.calc_K_s(K_accel_rotor, AR_rotor, beta_2m, beta_3m)
                            K_TET_rotor, N_rotor, c_true_rotor, c_a_rotor, throat_opening_rotor = aerodynamic_losses.trailing_edge_losses_r
                            K_rotor = K_p_rotor + K_s_rotor + K_TET_rotor
                            pitch_rotor = pitch_chord_ratio_rotor*c_true_rotor

                            # EFFICIENCY CALCULATIONS DESIGN POINT
                            eta_tt = aerodynamic_losses.efficiency_calculations(K_stator, K_rotor, M_2, M_3_rel, C_2, V_3)
                            delta_n, eta_final = aerodynamic_losses.efficiency_final(eta_tt, h, beta_3m, r_tip, r_meanline)

                            # OFF DESIGN LOSSES
                            eta_tt_od, delta_n_od, eta_final_od = aerodynamic_losses.losses_off_design(K_p_rotor, K_s_rotor, K_stator, K_1_

                            # EFFICIENCY CALCULATIONS OFF DESIGN
                            zxs = verify_zweifel_stator(r_hub_stator, h_vane_mean, N_stator, c_a_stator ,-10, alpha_2)
                            zxr = verify_zweifel_rotor(r_hub, h, N_rotor, c_a_rotor , beta_2m, beta_3m)

                            # WORK CHECKS
                            work_Part_A = c_p_gas * 1000 * (T_01-T_03)
                            work_check_cw = U * (C_w_2 + C_w_3)
                            work_check_vw = U * (V_w_2 + V_w_3)

                            if eta_final_od > 0 and zxs < 0.89 and zxr < 0.94: # This is to filter the abnormal behavior with the graph

                                # ==============================================
                                # START OF LOOP 2
                                # THIS LOOP PROVIDES A NEW DESIGN POINT AND ALSO UPDATES THE VALUES FOR ROTOR INLET AND OUTLET RELATIVE FLO
                                # Change the incidence angle
                                beta_2 = -1*incidence + beta_2m
                                beta_3 = beta_3m
                                incidence_off = incidence_2 - incidence

                                # OFF DESIGN LOSSES
                                eta_tt_od_new, delta_n_od_new, eta_final_od_new = aerodynamic_losses.losses_off_design(K_p_rotor, K_s_rotor

                                if eta_final_od_new > 0 and N_rotor < 50 and eta_final_od_new > 83:

                                    eta_final = eta_final/100
                                    eta_final_opt = 1/eta_final
                                    eta_final_od = eta_final_od/100
                                    eta_final_od_new = eta_final_od_new/100

                                    delta_eta_optimize = eta_final - eta_final_od_new
                                    data_scratch.append((zxs, zxr))
                                    data_meanline.append((j, l ,flow_coefficient_2, flow_coefficient_3, omega, m,    # Non dimensional para
                                                        T_01, T_1, P_01, P_1, rho_1, T_02, T_2, P_02, P_02_rel, P_2, rho_2, T_03, T_3, P_03
                                                        M_1, C_1, C_a_1, C_w_1, -10, M_2, C_2, C_a_2,  C_w_2, alpha_2, M_2_rel, V_2, V_w_2,
                                    # alpha_2_hub, alpha_3_hub, beta_2_hub, beta_3_hub, U_hub, V_2_hub, C_2_hub, M_2_rel_hub, M_2_hub, reac
                                    # alpha_2_tip, alpha_3_tip, beta_2_tip, beta_3_tip, U_tip, V_2_tip, C_2_tip, M_2_rel_tip, M_2_tip, M_3_
                                    data_root_hub.append((reaction_hub, h, r_tip, U_tip, r_hub, r_tip_stator, r_hub_stator, h_vane_mean,
                                                      alpha_2_hub, alpha_3_hub, beta_2_hub, beta_3_hub, U_hub, V_2_hub, C_2_hub, M_2_rel_hu
                                                      alpha_2_tip, alpha_3_tip, beta_2_tip, beta_3_tip, U_tip, V_2_tip, C_2_tip, M_2_rel_ti

                                    data_meanline_losses.append((c_true_stator, c_a_stator, stagger_angle_stator,K_p_stator, K_s_stator, K_

                                    data_off_design.append((T_3_od, rho_3_od, P_3_od, alpha_3_od, beta_2_od, flow_coeff_2_od, incidence_2, 

                                    data_efficiency.append((eta_tt, eta_final, eta_final_od, eta_final_od_new, eta_final_opt, delta_eta_opt



    data_scratch = pd.DataFrame(data_scratch, columns=['zweifel_stator_NEW', 'zweifel_rotor_NEW'])
    data_meanline = pd.DataFrame(data_meanline, columns=['stage_loading', 'reaction_meanline', 'flow_coefficient_2', 'flow_coefficient_3', 
    data_root_hub = pd.DataFrame(data_root_hub, columns=['reaction_hub', 'h', 'r_tip', 'U_tip','r_hub', 'r_tip_stator', 'r_hub_stator', 'h_
                                                         'alpha_2_hub', 'alpha_3_hub', 'beta_2_hub', 'beta_3_hub', 'U_hub', 'V_2_hub', 'C_2
                                                         'alpha_2_tip', 'alpha_3_tip', 'beta_2_tip', 'beta_3_tip', 'U_tip', 'V_2_tip', 'C_2
    data_meanline_losses = pd.DataFrame(data_meanline_losses, columns=['c_true_stator', 'c_a_stator','stagger_angle_stator','K_p_stator', '
    data_off_design = pd.DataFrame(data_off_design, columns=['T_3_od', 'rho_3_od', 'P_3_od', 'alpha_3_od', 'beta_2_od', 'flow_coeff_2_od', 
    data_efficiency = pd.DataFrame(data_efficiency, columns=['eta_tt', 'eta_final', 'eta_final_od', 'eta_final_od_new', 'eta_final_opt','de

    return data_meanline, data_root_hub, data_meanline_losses, data_off_design, data_efficiency, data_scratch
```

run int main

```
data_meanline, data_root_hub, data_meanline_losses, data_off_design, data_efficiency, data_scratch = int_main(range_mach_exit, range_stage_
```

Define Optimization Parameters

```
data_efficiency['delta_eta_optimize_normalized'] = (data_efficiency['delta_eta_optimize'] - data_efficiency['delta_eta_optimize'].min()) /
data_meanline_losses['N_rotor_normalized'] = (data_meanline_losses['N_rotor'] - data_meanline_losses['N_rotor'].min()) / (data_meanline_los
data_meanline['AN_squared_normalized'] = (data_meanline['AN_squared'] - data_meanline['AN_squared'].min()) / (data_meanline['AN_squared'].m
data_efficiency['eta_opt'] = 1/data_efficiency['eta_final']
data_efficiency['eta_opt_normalized'] = (data_efficiency['eta_opt'] - data_efficiency['eta_opt'].min()) / (data_efficiency['eta_opt'].max()
```

```
# OPTIMIZATION FUNCTION
data_efficiency['func_optimize'] = 0.4 * (data_efficiency['eta_opt_normalized']) + 0.3 * (data_meanline_losses['N_rotor_normalized']) + 0.2
```

Write results to file

```
data_meanline.to_csv('_outputs/output_data_meanline.csv', index=False)
data_root_hub.to_csv('_outputs/output_data_root_hub.csv', index=False)
data_meanline_losses.to_csv('_outputs/output_data_meanline_losses.csv', index=False)
data_off_design.to_csv('_outputs/output_data_off_design.csv', index=False)
data_efficiency.to_csv('_outputs/output_data_efficiency.csv', index=False)
data_scratch.to_csv('_outputs/output_data_scratch.csv', index=False)
```